CISC 322/326 Assignment 1

Conceptual Architecture of Kodi

Group 19 – Mystery Inc.

October 22, 2023

Authors

Brian Cabingan | 20bjc9@queensu.ca
Ethan Fighel | 20eaf4@queensu.ca
Zachary Kizell | 20zk16@queensu.ca
Daniel Madan | 18djm10@queensu.ca
Brett Morris | 19bcm2@queensu.ca
Daniel Solomon | 21ds28@queensu.ca

Table of Contents

Abstract

This report will explore the conceptual architecture of Kodi, a versatile and open-source media center application that has evolved from its roots as the Xbox Media Player to its current status as a comprehensive multimedia solution. Kodi's architectural style, known as Layered Architecture, is explored in detail, highlighting its suitability for managing the diverse aspects of a multimedia application independently. The four layers—Client, Presentation, Business, and Data—efficiently handle user interactions, content access, data conversions, and metadata tracking.This design allows for the independent management of different aspects of the application, facilitating the addition of new features and extensions without disrupting existing components.

Kodi also offers a range of external interfaces, including the Addons Framework for developers and the Python Scripts Interpreter for user customization, supported by open APIs for seamless platform communication. Metadata management plays a vital role in organizing media content by extracting relevant information from files or online sources. Kodi's versatile media handling capabilities cover a wide array of formats, supported by a core video player based on FFmpeg. Its open-source nature encourages user customization and community-driven development. The report includes practical use cases, highlights system evolution through community collaboration, discusses naming conventions for clarity, and shares valuable lessons learned about modular and extensible design principles.

Introduction and Overview

As digital entertainment continues to evolve and become a bigger part of our daily lives, media centers have become essential tools for anyone who enjoys consuming many different forms of digital media. Among the numerous media centers available is Kodi, which stands out

due to its versatility and ability to handle almost any type of digital media file you give it. It empowers users to take control of their media experiences, providing a customizable and extensible platform that caters to individual preferences. Moreover, the open-source community surrounding Kodi has contributed to its success by continually enhancing its features and expanding its capabilities, allowing the application to live up to its full potential.

Originally named XBMP (Xbox Media Player) and soon after XBMC (Xbox Media Center), this multimedia tool that now supports dozens of file formats was initially designed to allow the Xbox to be more than just another gaming console. The creators wanted Xbox users to be able to run other types of media such as videos, music, and images, all while having a user-friendly and effortless experience. Over time, XBMC became a widely used app for enhancing the Xbox, and so the creators decided to turn it into a cross-platform application so that it could be accessible to many more people. This eventually led to their decision to change the name to Kodi, a shorter, simpler name that made the app sound like it was meant for more than just a single gaming console.

The purpose of this report is to explore the conceptual architecture that forms the backbone of Kodi and understand the reasoning behind it, as well as the pros and cons of using the architectural style that was chosen. The developers behind Kodi employed a layered architectural style, which is typically used for systems that involve classes of services that can be organized in a hierarchical manner. This style was quite suitable for a system like Kodi, as layered styles are great for handling different aspects of a versatile application independently - user interface, media handling, and data management are all aspects that can be maintained on their own while still being able to efficiently interact with one another. Layered styles also allow for seamless scalability of a system; new features or extensions can be added without disrupting existing layers. The structure behind this system will be explored in more detail further in this report, and we will discuss the reasons behind why we think the developers ultimately decided to choose this style.

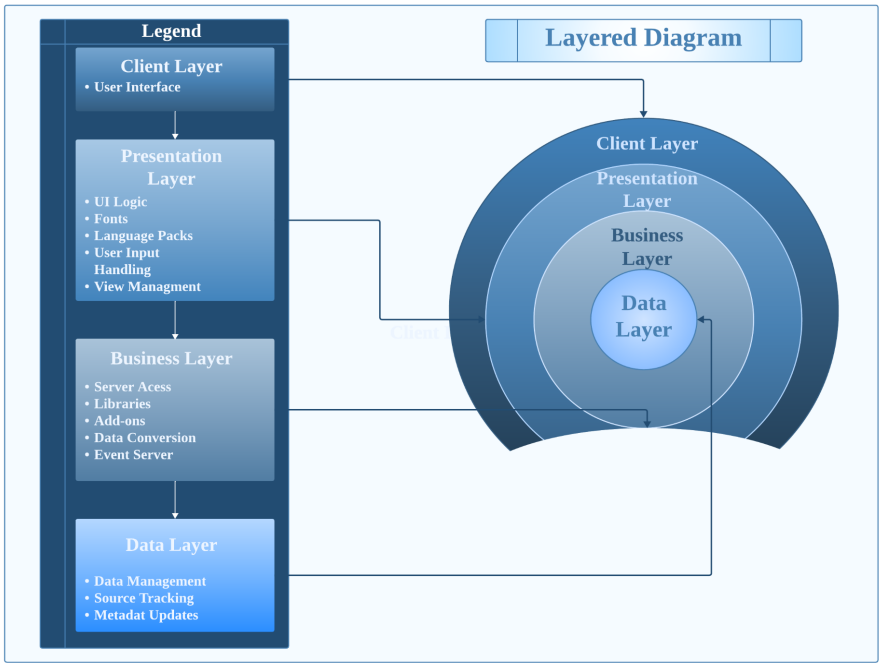
Additionally, Kodi is a fully open-source application, meaning the application's source code is freely accessible and can be modified, extended, and customized by a global community of developers and enthusiasts. This open-source format is not only a great environment for innovation and collaboration but also empowers users to take control of their media center, which is not usually possible with most other multimedia applications. At the time of writing this report, the Kodi GitHub repository has over 800 contributors, which goes to show how much work is put into an application of this size and quality.

In the following sections of this report, we will delve further into the architectural details of Kodi. We will analyze the layered architectural style, examining how it manages and complements the diverse media types and ensures a user-friendly experience. The open-source community's major role in Kodi's development will also be examined, and we will look at how this development style came to bring us the media center we have today.

Architecture

Parts & Components

Kodi employs the Layered Architecture Style, evident in its foundational core layer serving as the backbone upon which all other components are constructed.. The Layered Architectural Style is typically used for applications that involve distinct classes or services that can be systematically structured in a hierarchical manner. Within this framework, components represent collections of procedures, and connectors are essentially procedure calls with restricted visibility. Each individual layer can only communicate with layers that are adjacent to itself (see diagram below). Kodi has four layers / components: Client, Presentation, Business and Data. The Client Layer is where the user accesses the application. Kodi is available on TV's, Android / iOS devices and on computers.



The Presentation Layer is responsible for the code that dictates how a user will interact with the program on the front end, containing files for fonts, language packs, user input, and view management. Next is the Business Layer, which is used to access external content through server access, libraries, and addons. This layer will then make necessary conversions so that the data can be displayed for the user. A major component of this layer is the EventServer, which was created to support input from a multitude of hardware platforms. Since direct support is known to decrease performance and stability, and is hard to maintain, the EventServer was created to simplify communication with Kodi. Finally, the Data Layer keeps track of and updates sources, files and metadata. (Kodi Wiki, 1)

Addons:

Kodi has several open APIs which allows developers to create addons to the system. The Addons Framework architecture and Addons Manager GUI client connects to a decentralized digital distribution platform which users can download addons from. Additionally, Kodi uses an integrated Python Scripts interpreter for addon extensions and allows users to create new functionality to the system. Addons are usually made by third party developers and can be published into Kodi's official repository. There are three different kinds of addons: System

addons, Official Repository addons, and Private Repository addons. System addons are installed by the Kodi system and can only be updated by the system. Official Repository addons are addons contained in the official Kodi repository. Private Repository addons are addons that the user installs and are loaded from a private repository. These addons are then further broken down into different types such as plugin sources, repository, and scraper addons, each serving a different function. (Chiappetta M, 1)

Video:

Kodi's video library, one of its metadata databases, organizes video content using information within the video files. This allows users to browse their videos using various categories. Kodi also uses a core video player called DVDPlayer which is based on FFmpeg for video playback. (Paul R, 1)

Music:

Another one of Kodi's metadata databases is its music library which allows the organization of a collection of music files for easier searching and creating playlists based on information stored in the music file. (Patowary K, 1)

Media Formats:

Kodi uses several internal systems to play various media formats such as a DVD-ROM drive, hard disk drive, Windows File-Sharing, and Network File System. Kodi can also use an internet connection to view internet videos found on such as those found on YouTube and Netflix. (Patowary K, 1)

Pictures:

The pictures menu allows users to view photos within the Kodi system. Kodi uses CxImage open source library code to handle digital image formations and while it does not use a database to store information like the video and music libraries, it is a simple to use file browser. (Kodi Wiki, 1)

Live Television:

Kodi has a feature that allows users to watch Live TV using EPG and DVR features with a Personal Video Recorder frontend GUI and optional addons for backends. The PVR backend can either be a DVR set-top box connected to the network or a PC with a digital video recorder software. The PVR addons can be added to the system depending on the users' preference such as addons for PVR software, direct LAN connections, or internet-based television providers. (Kodi Wiki, 1)

System Evolution

Kodi is a fully open source project where all fixes are made by the community. Extensive documentation for potential developers has been created using Doxyfile that outlines a set of best practices. Kodi makes use of GitHub Issues to track bugs submitted by the community. Then, community developers will create their own fixes and create a pull request, so that their fixes can be reviewed before being merged into the main branch. Devs have access to nightly builds to

test, and as the builds become more stable public betas and release candidate builds are created for further testing.

The layered architecture style is helpful for evolving the system as a whole. Since it is structured around creating layers of abstraction, changes to one layer will affect other layers. Additionally, different implementations of the same layer can be used interchangeably, which makes rewriting or updating components easier. This also makes it easier to test components, since other layers can be mocked, allowing devs to find issues more easily.
A core guideline for the development of Kodi was that it "should still compile and run if a non-essential module/library is removed." (Quote from Kodi wiki). This is helpful because if there is a failure the software can still be compiled, used, and tested. Using modules makes the code easier to maintain and scale. Kodi's features are relatively independent of each other, functions do not need to be developed multiple times, since they are abstracted into different layers. The independence of modules also makes it much easier to maintain and expand features for the system. (Dreef et al, 1)

Kodi started off as an open-source media player for the Xbox called the Xbox Media Player which allowed users to access pictures, movie files and play music. The original project was founded by Duo Egaq and Albert Griscti-Soler who were, at the time, working on their own separate players until they decided to merge during the Xbox Media Player beta 5. Complaints were made after the release of beta 6 due to the developers not releasing the source code as they were using code that was under the LGPL license. Specifically, they used code from FFmpeg, which is a collection of libraries and programs, and Xvid, which was a video codec library. The two closed down the project and released the source code for beta 6 on October 15, 2002. (Kodi Wiki, 1)

A software developer nicknamed Frodo joined the Xbox Media Player team in November 2002 and merged his project, Yet Another Media Player, with the XBoxMediaPlayer to create the Xbox Media Player 2.0. They released the source code for this project on December 14, 2002 which showed that it was still using FFmpeg and Xvid code, but otherwise it was a complete rewrite. Later on, the team began releasing newer versions that included bug fixes as well as additional features. Version 2.1 included AC3 5.1 output, volume normalizer/amplification and an additional post processing filter. Version 2.2 included separate national language files, streaming media from windows file shares, audio-playlist, the ability to play media from ISO9660-Mode1 CDs, dashboard mode which allowed users to launch other Xbox applications/games, and Windows DLL support for WMV 7,8, and 9. However, this project stopped development on December 13, 2003. (Kodi Wiki, 1)

The successor to the Xbox Media Player 2.0 was the Xbox Media Center (XBMC) which was released on June 29, 2004. XBMC allowed for embedded Python code which allowed users to create scripts to interact with the XBMC environment. New releases allowed for more support for new media types, file types, and container formats. The release of version 2.0.0 on September 29, 2006 showed a brand new player interface with support for multiple players with the addition of RAR and ZIP archive support. On May 29, 2007, the development team was interested in porting XBMC to the Linux operating system so they put out a call for developers. Finally, in July 2014, the project was renamed from XBMC to Kodi 14. (Kodi Wiki, 1)

Naming Convention

In Kodi there are two options for organizing movies within the source folder. The first is called "Movie Folders," where each movie has its dedicated folder within the source directory, which is the recommended method. The second option is "Flat Folder," where all movie files are directly saved in the source folder without any subfolders(). Within Kodi's codebase, it follows "Camelcase" notation for functions and "PascalCase" for class names. Additionally comments and naming consistency in important to note. Variables, functions, classes and interfaces should have names correlating to their designated purposes (Kodi Wiki,1).

Division of Developer Responsibilities

It should be noted that despite being open source, where anyone can make a contribution, Kodi's development team does not have a flat structure. A small group of devs who have provided significant contributions to the development of Kodi have been given permission to review and approve or deny pull requests. They often work closely with users, who they rely upon to report bugs and propose new functionality. Additionally, Kodi does have board members who are responsible for task management for developers. These board members are usually former developers, all of whom have been elected by the community. Many of these board members are still very active developers on top of their responsibilities as board members. (Dreef et. al, 1)

External Interfaces

Presentation Layer:

- User Input Module: The inputs sent between the user's device and the plug-ins on the Business Layer. (Kodi Wiki, 1)
- Windows Manager Module: It uses the libraries from the operating system to help with window management. (Kodi Wiki, 1)
- GUI Elements: Provides the user with the options to consider and select options that are sent to the system. (Kodi Wiki, 1)
- Translation Module: It receives the request from a user to display a certain language onto Kodi. (Kodi Wiki, 1)
- Audio Manager Module: Uses the device's audio device to output sound, changes the device's volume and output mode. (Kodi Wiki, 1)
- Fonts Module: Stores GUI appearance settings and receives messages from the user to toggle the settings. (Kodi Wiki, 1)

Business Layer:

- Python Module: Allows the user to create their own addons in Python, otherwise interacts with external services through Python to give the user access to addons in Kodi. (Kodi Wiki, 1)
- Web Server Module: Multiple users can access and manage the same media content (if they are connected to the same local network) through the web or other external applications. (Kodi Wiki, 1)
- Player Core Module: It decodes multi-media files so that the hardware or software required for their playback can run them. (Kodi Wiki, 1)

Data Layer:

- Sources Elements: Receives and stores various types of content sources such as storage drives, network protocols and streaming devices to be selected by the user. (Kodi Wiki, 1)
- Views Elements: Displays the data on their existing media library to the user in a visually clear fashion. (Kodi Wiki, 1)
- Metadata Elements: Reads the data of imported media such as title, genre, language and more. Information is derived from metadata from the imported files or found through online databases. (Kodi Wiki, 1)

Use Cases

Below are two use cases that the system anticipates from any potential users:

Use Case 1:

As a user, I would like to be able to access game files in my local file system on my PC so that I can play those games using the plugins I can download onto Kodi from the Internet.

The user accesses Kodi through the client layer which in this case is the PC. This boots up the Presentation Layer's view management software so the user can see all their files that have been linked up to Kodi through the Data Layer that interacts with the actual file system.

In this use case, let's assume that no files have been linked up to Kodi, and that the user wants to access game files. The first step would be to navigate to the games section of Kodi, and search for a folder using the Data Layer that has the games you want to play. Then, that folder is linked up and you can directly access anything in that folder by just navigating through it on Kodi, which involves constant source tracking to make sure we specify all the root directories that are then kept track of by the data management component.

Once you find and select the game you want to play using the directory navigation provided by the view management component, it will search for plugins on the web using the Addons Framework in the Business Layer and return results based on what kind of game file you are planning to run. Once you choose one, you are able to set up any external devices like joysticks or other computer settings through Kodi using the Event Server to adapt to the requirements of the plugin, and then you can proceed to play the game! (Kodi Wiki, 1)

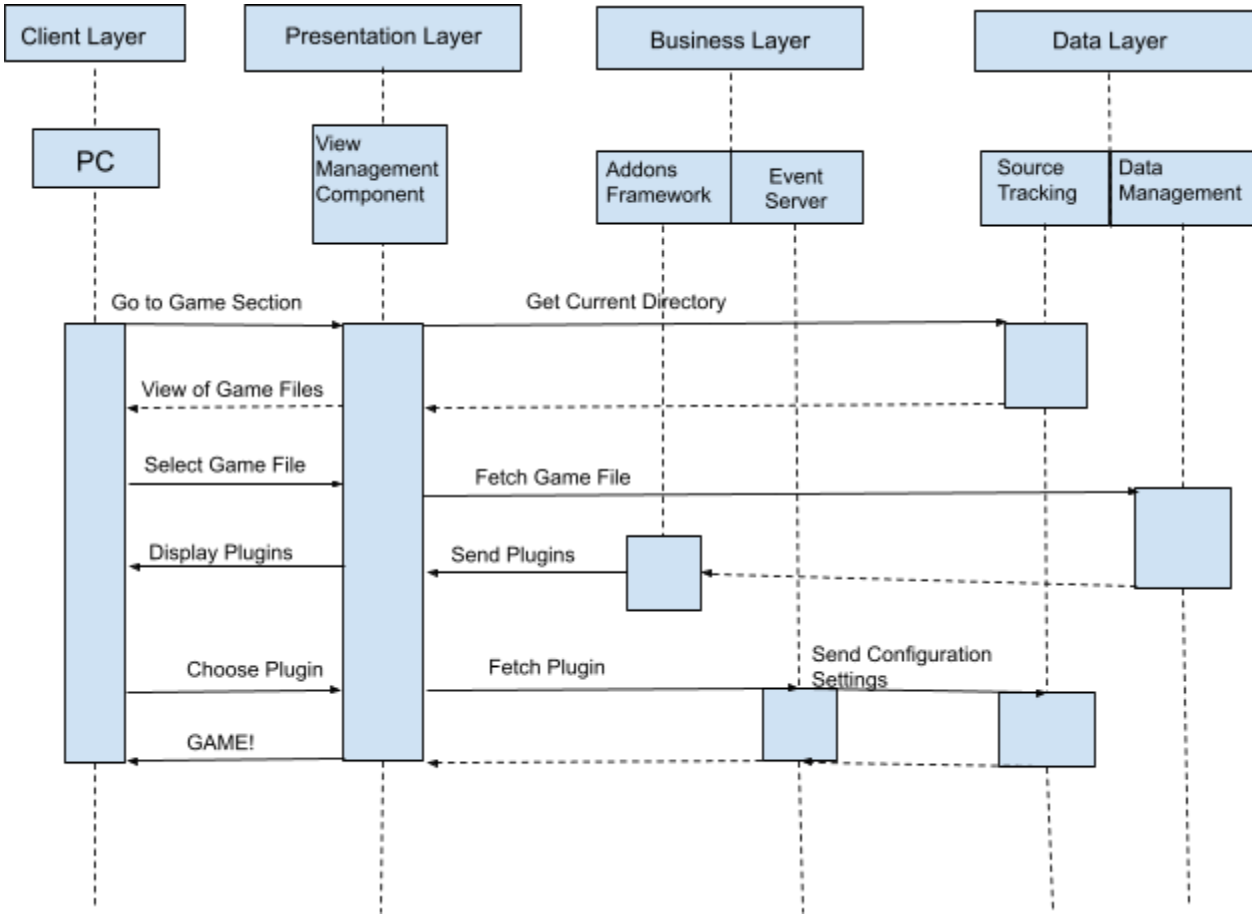


Figure 1: Sequence diagram of use case 1

Use Case 2:

A user wants to search for a specific movie and watch it on their smartphone.

The user starts by opening the program on their smartphone as part of the Client Layer and accessing the GUI Elements as part of the Presentation Layer. Selecting the “TV” element opens the Web Server Module sections on the Business Layer, which draws a list of all of the previously added TV shows/movies from the Library in the Data Layer. From here, the user can either select a previously added video or add a new one.

If the video they are looking for is already there, they can select it from GUI Elements, which takes them to the Player Core Module in the Business Layer. There, the DVDPlayer component broadcasts the video to the user’s screen.

If the video they are looking for is not there, they will need to click on the button that allows them to browse their computer for TV show/movie files to upload. The user can upload files from one of the Sources components in the Data Layer, which will add that file to the Library component in Views. From there, the player can select that file using the GUI Elements

in the Presentation Layer and view the movie from the DVDPlayer component in the Business Layer. (Kodi Wiki, 1)

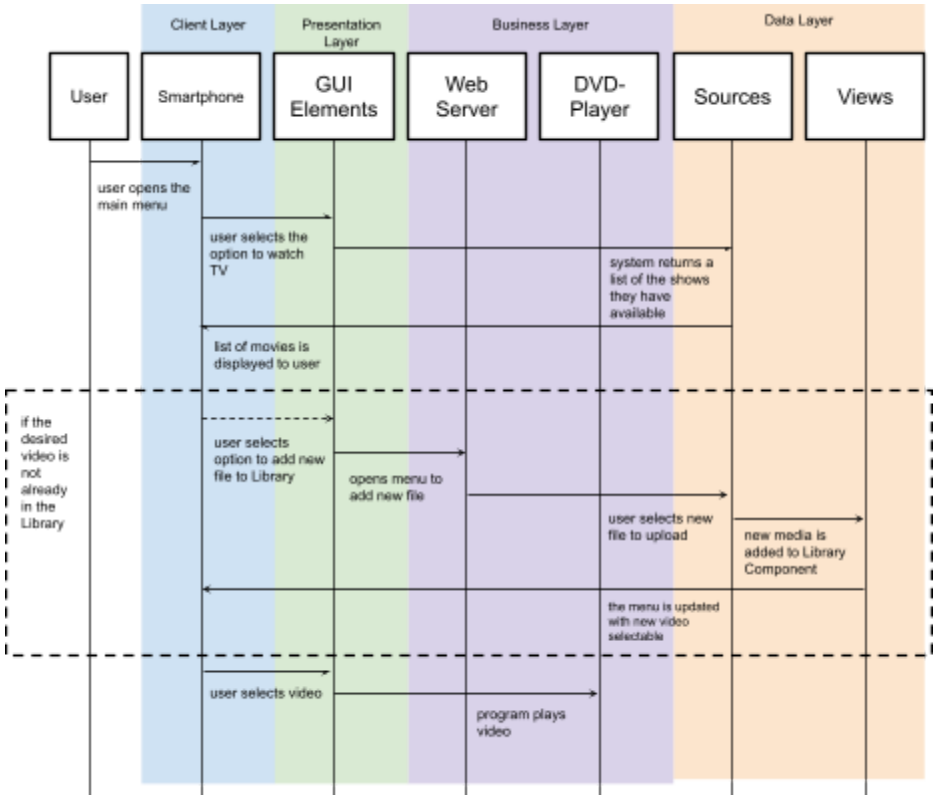


Figure 2: Sequence diagram of use case 2

Data Dictionary

Electronic programming guides (EPG) - Menu-based systems that provide users of media applications, such as television and radio, with an updated menu which displays scheduling information for current and upcoming broadcast programming

Digital Video Recorder (DVR) - An electronic device that records video in a digital format to a local or networked mass storage device such as a sordid state drive, USB flash drive or a disk drive

Conclusions

In summation, Kodi has solidified its position as a versatile and enduring media center application amidst the current technological revolution. With a layered architecture encompassing Client, Presentation, Business, and Data layers, Kodi efficiently manages user

interactions, data access, and content delivery. This modular design not only ensures flexibility and ease of maintenance, but also allows Kodi to evolve continuously. From its origins as Xbox Media Player to its transition to Kodi, the project has grown, added features, and garnered a dedicated community of users and developers. Kodi's success reflects the importance of adaptable architecture and community-driven development practices in remaining relevant in the dynamic world of digital media.

As discussed above, the layered architectural style employed by Kodi plays a key role in its evolution and ongoing success. This style allows for systematic organization, modularity, and scalability, which are essential for an application of this magnitude. The independence of layers and modules simplifies maintenance and facilitates expansion as well. The collaborative efforts of both developers and users have propelled Kodi's evolution, ensuring it remains a top of the line multimedia platform, accessible to a global audience. All of the features and architectural components that make Kodi what it is position it as a competent and innovative digital media platform, continually adapting to meet the evolving needs of its user base.

Lessons Learned

Throughout the completion of this report on the architecture of Kodi, there are a couple of noteworthy insights that have surfaced. These insights not only serve as valuable takeaways from this report but also offer a broader perspective on software development and open-source projects in general.

Something we learned early on is that it’s evident that the choice of architectural style plays a crucial role in a project's long-term success and efficiency. In the case of Kodi, the choice of a layered architecture style greatly contributed to how adaptable and scalable the system was. If an architecture style that did not match well with the requirements of Kodi was used instead of the layered style, enhancing and scaling it could have taken much more time and resources, and its development as a whole would not have been nearly as smooth. It's definitely worth noting that the choice of architecture can profoundly impact the project's capacity to evolve and meet changing requirements.

Looking back at the development process, it becomes apparent that defined responsibilities among developers are a vital aspect in ensuring a smooth development process, especially for a large-scale application like Kodi. Kodi's division of responsibilities, with a small group of core developers and elected board members, ensures efficient task management and collaboration. An organizational structure like this is essential for larger open-source projects to maintain focus and direction. Clear roles and responsibilities can prevent conflicts, streamline decision-making, and ensure the project's longevity.

References

Dreef et al. (2015, April 23). *Architecting Software to Keep the Lazy Ones On the Couch*. GitHub. https://delftswa.github.io/chapters/kodi/

Hassan, A. (n.d.). *Module 03: Architecture Styles*. CISC 326 OnQ. https://onq.queensu.ca/d2l/le/content/820310/viewContent/4911384/View

Kodi Foundation. (n.d.-a). Kodi Wiki. https://kodi.wiki/view/Main_Page

Kodi Foundation. (n.d.-b). *Team Kodi*. GitHub. https://github.com/xbmc

Chiappetta, M. (2013, March 21). HTPC showdown: Which front-end interface is best?. TechHive. https://web.archive.org/web/20141127211232/http://www.techhive.com/article/2031217/htpc-showdown-which-front-end-interface-is-best-.html

Paul , R. (2009, December 29). XBMC 9.11 makes your open source home theater look shinier. Ars Technica. https://web.archive.org/web/20120416191227/http://arstechnica.com/open-source/reviews/2009/12/xbmc-911-makes-your-open-source-home-theater-look-shinier.ars

Patowary, K. (2009, August 28). XBMC is the Best Media Center application. period. Instant Fundas. https://www.instantfundas.com/2009/08/xbmc-is-best-media-center-application.html